ELSEVIER

Parallel Computing 26 (2000) 243–266

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Massively parallel computing using commodity components

Ron Brightwell[a,*], Lee Ann Fisk[a], David S. Greenberg[b,1], Tramm Hudson[a], Mike Levenhagen[a], Arthur B. Maccabe[c,2], Rolf Riesen[a]

[a] *Scalable Computing Systems, Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM 87185-1110, USA*

[b] *IDA/CCS, 17100 Science Drive, Bowie, MD 20715-4300, USA*

[c] *Department of Computer Science, University of New Mexico, FEC 313, Albuquerque, NM 87131, USA*

Received 10 November 1998; received in revised form 7 June 1999

Abstract

The Computational Plant (Cplant) project at Sandia National Laboratories is developing a large-scale, massively parallel computing resource from a cluster of commodity computing and networking components. We are combining the benefits of commodity cluster computing with our expertise in designing, developing, using, and maintaining large-scale, massively parallel processing (MPP) machines. In this paper, we present the design goals of the cluster and an approach to developing a commodity-based computational resource capable of delivering performance comparable to production-level MPP machines. We provide a description of the hardware components of a 96-node Phase I prototype machine and discuss the experiences with the prototype that led to the hardware choices for a 400-node Phase II production machine. We give a detailed description of the management and runtime software components of the cluster and offer computational performance data as well as performance measurements of functions that are critical to the management of large systems.

Keywords: Massively parallel; Workstation cluster; Beowulf; Distributed memory; Message passing; MPI

* Corresponding author. Tel.: +505-845-7432; fax: +505-845-7442.

E-mail addresses: cplant-dev-nm@cs.sandia.gov (R. Brightwell), dsg@super.org (D.S. Greenberg), maccabe@cs.unm.edu (A.B. Maccabe).

[1] Contributions to this paper represent work done at Sandia National Labs.

[2] Tel.: +505-277-6504; fax: +505-277-6927. Supported in part by NSF CISE Research Infrastructure Award CDA-9503064.

0167-8191/00/$ - see front matter
PII: S0167-8191(99)00104-0

1. Introduction

Using tightly coupled clusters of commodity off-the-shelf (COTS) personal computers (PCs) to build a dedicated parallel computing resource has become popular due to cost/performance benefits. Small-scale clusters have shown performance comparable to small versions of massively parallel processing (MPP) systems [26]. While the initial steps toward commodity cluster supercomputing have shown promise, many obstacles remain for the development of large-scale clusters with the compute and I/O power required to compete directly with large commercial MPP systems.

Several efforts, both in industry and in research institutions, are attempting to build scalable clusters composed of commodity computing and networking hardware. While these projects address many of the issues relevant to the size and capability of the targeted systems, many issues and complexities must be addressed to scale these types of systems to reach Sandia's current one trillion floating point operations per second (TeraFLOPS) level of compute performance. In this paper, we present Sandia's approach to constructing a large-scale, MPP machine constructed from commodity components.

In the following section, we present an overview of the Sandia/Intel TeraFLOPS (TFLOPS) machine, an MPP representative of the computing capacity and capability required by Sandia's applications. Section 3 summarizes the shortcomings of previous commodity cluster projects, and Section 4 discusses the concept and design of Sandia's approach to building a scalable, high-performance, production quality cluster. Sections 5 and 6 present an overview of the hardware components of the prototype and production clusters. Section 7 describes the software components of the diagnostic and management systems, and Section 8 details the software components that compose the scalable runtime system. Performance results of parallel benchmark codes are presented in Section 9. We conclude with a discussion of future work and a summary of the key contributions of this research.

2. Sandia/Intel TFLOPS machine

The Sandia/Intel TFLOPS machine is part of the Department of Energy's (DOE) Accelerated Strategic Computing Initiative (ASCI). TFLOPS is the culmination of more than 10 years of research and development in massively parallel, high-performance, distributed memory computing.

2.1. Hardware

The Sandia/Intel TFLOPS machine is composed of more than 9000 Pentium II Xeon processors connected by a network capable of delivering 800 MB/s bidirectional bandwidth (see [19] for details). The machine has a peak performance of 3.2 TeraFLOPS, and as of November 1999, holds the world record for compute

performance [16], demonstrating 2.37 TeraFLOPS on the LINPACK [6] benchmark. This level of performance is required to solve the types of problems that are critical to Sandia and the ASCI program [1].

2.2. Software

The high-performance operating system on the TFLOPS compute nodes was designed and developed by Sandia and the University of New Mexico [23] and ported to the TFLOPS by Intel. The design of this lightweight kernel stems from earlier experiences in providing a high-performance operating system optimized for distributed memory, message passing MPPs [12]. A key component of the system software is a high-performance message passing layer called Portals [21]. Portals are data structures within the address space of an application that tell the kernel how message operations should be processed. Portals were one of the earliest attempts to establish a zero-copy application bypass mechanism for message passing. Using Portals, messages are delivered directly to the application with no need for application-level processing. In particular, Portals does not require an application-level thread that continually polls the network interface to process messages. Such threads are typically used in other systems to deliver messages to the correct place based on fields in the message. Commodity networking technology has recently begun to realize the benefits of decoupling the network from the application processor, with emerging technologies such as the Virtual Interface Architecture (VIA) [5] and Scheduled Transfer (ST) [25]. [3]

2.3. Partition model

The TFLOPS employs a partition model of resources [8], where each partition provides access to a specialized resource. Fig. 1 illustrates a typical logical configuration. The largest portion of the machine is the compute partition, which is dedicated to delivering processor cycles and interprocessor communications to applications and ideally runs a lightweight operating system. The service partition provides a full UNIX environment where users log in to access the compute partition. Nodes in this partition provide the familiar UNIX shells, tools, and utilities. Parallel jobs are launched from the service partition into the compute partition.

The TFLOPS is managed as a space-shared resource. Nodes are allocated to a specific job and user. Once allocated, a node can only be used by that user until the job terminates. The runtime environment supports multiple processes per node, but this feature is rarely used in practice. Typically a job will need all of the compute,

[3] Portals support an application bypass in the sense that when the Portals code is interpreted on a coprocessor, messages can be delivered to the application without interrupting the execution of the application. However, unlike current operating system bypass strategies, the Portals implementation requires full address translation support.

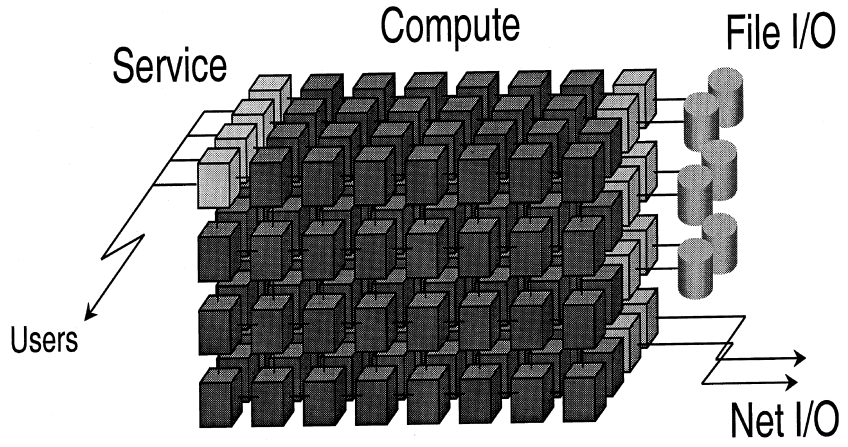

Fig. 1. Conceptual partition model.

memory, and communication resources a node can provide. If more compute power is needed, nodes are added to the compute partition.

2.4. Maintenance and diagnostic system

The TFLOPS has a separate subsystem for maintaining and diagnosing the machine. The reliability, availability, and serviceability (RAS) system consists of a separate network of processors that monitor the health of the machine. RAS is used to identify components of the machine that are failing or have failed, configure the nodes out of the system, and then reconfigure the nodes back into the system after replacement or repair. For the 1998 calendar year, the TFLOPS incurred only seven hardware failures that caused the machine to become unusable, and the availability rating was 98% (disregarding scheduled maintenance). Such an RAS system is critical for large-scale production systems.

2.5. Limitations

The TFLOPS has proven to be one of the more technically successful efforts in massively parallel, high-performance computing. However, large MPP systems have drawbacks. Future high-performance computing platforms of this magnitude need to address the following issues.

Custom hardware components are quickly superseded by commodity components. The TFLOPS is based on commodity processor technology, but many of the hardware components were designed and built specifically for the TFLOPS. In particular, the network and RAS systems were designed specifically for the TFLOPS. The use of custom parts significantly increase the cost of the machine, in money as well as time. The performance of the original 200 MHz Pentium Pro processors in

the TFLOPS was quickly overcome by that of the Pentium II and Alpha processors. Commodity networking and I/O bus technology will soon allow for networking performance to reach the level of the TFLOPS network. In addition, the procurement process for large-scale machines is typically on the order of several months. The cost of some components can drop dramatically between the time the contract is finalized and the machine is delivered.

Volume vendors are not the best organizations to create niche products. High-performance, massively parallel scientific computing continues to be an extremely small market for volume hardware and software vendors. Companies that target the mass market may be unwilling and/or unable to address the needs of the scientific community, primarily due to lack of potential revenue.

Large system scalability requires specialized knowledge and research. The success of the TFLOPS machine was the result of 10 yr of research and development. Much of the knowledge of how to build systems that can scale to thousands of nodes has not yet permeated the high-performance computing community as a whole, and much research remains to be done.

Large-scale systems must grow in size and capability. Otherwise, they risk becoming obsolete before becoming fully functional. Hardware and software technology in the current arena of high-performance computing is evolving rapidly. Large-scale systems must be able to adapt and evolve with these new technologies in order to remain a valuable resource.

The applications that require high levels of compute performance will continue to grow in size, variety, and complexity, and large-scale systems will have to evolve with these changes. Systems must be able to continually adapt to the environments and capabilities the applications require.

3. Commodity clusters

The ability of the TFLOPS to scale to more than 4000 nodes has been cited as an important factor in establishing the feasibility of high-performance commodity cluster computing [24]. There are numerous projects attempting to construct commodity-based clusters as a low-cost alternative to commercial parallel computing systems (see [4] for several such projects). These projects have made much progress in establishing a foundation upon which small- and medium-scale clusters can be based. However, the current state of cluster technology does not support scaling to the level of compute performance, usability, and reliability of large commercial MPP systems like the TFLOPS.

In order to build workstation clusters with thousands of nodes, scalability of *all* system components is critical. Dependence on non-scalable technologies must be bounded so the limits of scalability are not approached. When bounding is not possible, the dependence on non-scalable technology must be eliminated. Network technologies such as Fast Ethernet, wide-area network protocols such as TCP/IP, and tools that are built on these protocols, such as Network File System (NFS) and remote shell (rsh) have inherent scalability limitations that must be addressed.

Scalability spans every aspect of a large-scale machine. For example, scalability must be provided to the application, not only through high-speed networking, but also through a scalable startup mechanism in which compute resources are allocated and processes are started on thousands of nodes.

The machine must also be managed and maintained in a scalable fashion. The complexity of administration should not increase significantly as the size of machine or the number of users increases. For large-scale clusters, efficient maintenance and management of the system is at least as important as efficient use of the system.

Usability of the machine is critical. Users should be able to interact with and use the machine without any specialized or intimate knowledge of the underlying components. That is, users should not be required to know the name of every node in the cluster. Nor should users be responsible for determining the allocation of the machine's resources.

Ideally, clusters should be able to support the same types of applications MPPs support. Current cluster systems support a small number of users and run a small subset of applications. In order to approach MPP systems, clusters must address the issues related to making a wide variety of applications run well.

4. Computational plant

The Computational Plant [20] (Cplant) project at Sandia National Laboratories addresses the above issues relevant to developing, using, and maintaining a massively parallel commodity-based cluster. It combines the benefits of commodity cluster computing with expertise in designing, developing, using, and maintaining large-scale, MPP machines. The goal is to provide a commodity-based, large-scale computing resource that not only meets the level of compute performance needed by Sandia's critical applications, but that also meets the levels of usability and reliability established by the TFLOPS. We have proposed a new model for the creation of high-end capability resources. Cplants are built on the design of the TFLOPS system, from which several key concepts can be derived:

- Large systems must have significant resources dedicated to system monitoring and system configuration.
- Large systems should be constructed from independent building blocks.
- Independent building blocks can be partitioned by changing a small number of system characteristics.
- Large systems should be partitioned into conceptually separate components that provide specialized functionality.
- Partitions can interact through a limited number of capabilities.

4.1. Conceptual design

In order for a Cplant to be integrated from components available from a variety of vendors, an architecture that is easily expandable and integratable is needed. The Cplant architecture consists of *scalable units* and a *support system*.

The definition of a scalable unit (SU) is intended to be as non-specific as possible in order to allow a variety of vendors to supply cost-effective components. The use of the partition model of resource provision [8] allows SUs to provide a variety of resources, such as service, compute, I/O, and network. Most of the Cplant resources are likely to be devoted to creating computational nodes, i.e. nodes which can service distributed memory programs and, at a minimum, run an MPI process. At least one SU must provide a service capability (i.e., a direct interface to users from which jobs can be started, monitored, and debugged). SUs can also provide specialized resources, such as enhanced secondary or tertiary storage and enhanced network capability. A specialized resource may appear as a service that can be called from an application code within a compute partition and/or be called via a command line argument by users through a service partition.

SUs must also respond to the management queries and configuration commands defined for the support system. For example, it is desirable that an SU accept commands for remote power-cycling, and to install or update any software on the system. Each SU should also be able to report status, such as a excessive memory errors. The physical boundary between the support system and the SU can be somewhat blurred. If SUs do not provide sufficient support for control and query, then the support system may be extended to include a system support station (SSS) that is "attached" to the SU. In this way, the support system grows in a scalable manner with the number of SUs, but the SUs are not required to include support system functions by design.

The primary purpose of the support system is to mold SUs together into a single system. Most of the functionality of the support system consists of querying and controlling SUs. Typically the support system will encompass a superset of the functionality of any single SU, since it must support all SUs. The support system has two logical pieces, a system support network and a (usually existing) local infrastructure.

The system support network provides the ability to configure, customize, monitor, maintain, and control the entire Cplant. The simplest hardware realization of a system support network might be a console and keyboard with physical connections to every component of the system. This solution, however, is not scalable. A minimal hardware realization must include a special component associated with each SU that serves as a proxy for all interaction with the SU, a system console that serves as the point of entry for system administrators, and a network connecting SU components to the console. In general, a broadcast medium is preferable. Fault tolerance through multiple paths in the network and multiple consoles is also desirable.

The following is an overview of some of the classes of service that the system support network must provide. Once attached to the system, it must be possible to configure an SU from the system console. It must be possible to monitor the health of all components in the system and to collect information about the current state of the system. An SU must be able to inform the system support network about the resources it provides.

The information collected above may be used in several ways. A system administrator must be able to access the system console and execute commands. The allocation of resources within the system will have to be managed by a distributed management system. System libraries, such as MPI, can be optimized if given detailed information about the system configuration. Tools such as debuggers and performance monitors could make use of some of the functionality of the system support network.

While access to the system support network by ordinary users is undesirable, certain parts of the system status may be of interest. For example, a user might wish to see a representation of available resources and who is using them.

In addition to the system support network, a local infrastructure must be available to the Cplant. Any computing resource not available at users' desktop is not really available. Some resources must provide services directly to the user, for interactive use or batch queue submission. These command interfaces require a network connection to the service partition. Most users will have workstations and access to shared file servers where code and data are stored. The service partition should also have a mechanism to access these files. Files created by applications will most likely need to be examined externally, and data may need to be transferred to external storage systems such as tape archiver (e.g., HPSS) or visualization servers.

5. Phase I – Prototype Cplant

During the Fall of 1997, Sandia designed and integrated a 96-node prototype Cplant. The primary goals were to validate the conceptual design and to create a reference SU for future additions to the cluster. In addition to these development goals, the prototype was to be a resource for application development. In order to ease the transition of users to the prototype, the familiar TFLOPS runtime environment was replicated so that application developers could interact with the cluster in the same manner as the TFLOPS. The following sections outline the hardware components of the prototype and some of the challenges these components presented.

5.1. SU

Fig. 2 shows the components of an SU for the prototype machine. Each SU consisted of two cabinets, each with eight nodes and a power controller. One cabinet contained the high-speed networking switches, which were laid in the floor of the cabinet. Each cabinet had eight incoming serial lines from the SSS-0, one to the serial port of each machine. The SSS-0 machines were also mounted in a cabinet, with only two SSS-0s per cabinet because of distance limitations discussed below.

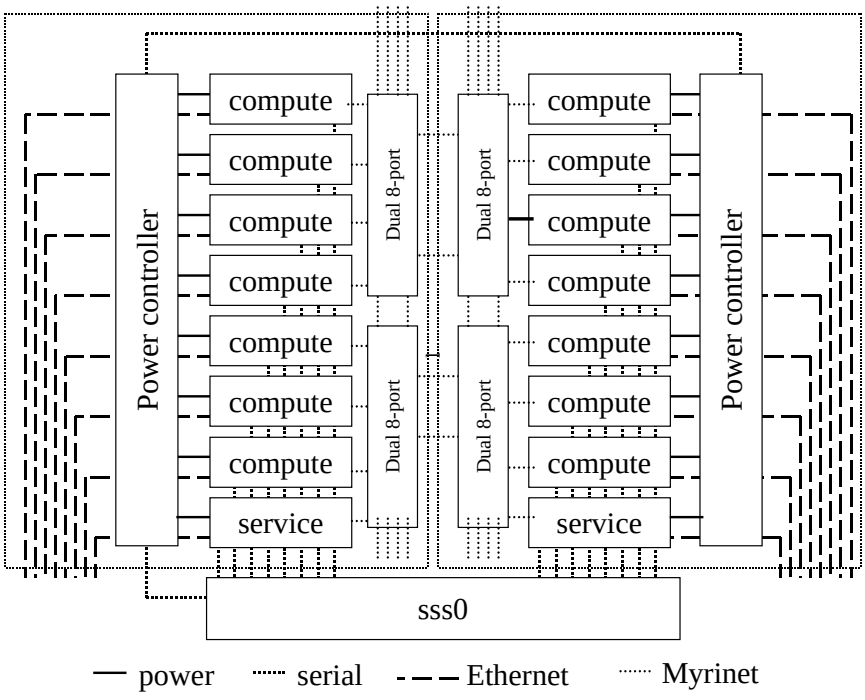

Fig. 2. Phase I scalable unit.

5.2. Hardware

The first step in constructing the prototype was to develop a set of requirements and evaluate possible candidates for different components. While other cluster projects have emphasized achieving low price/performance levels, this goal was not paramount for the prototype. An effort was made to keep costs as low as possible, but, in some instances, solutions that exhibited more scalability or had higher absolute performance were chosen over solutions that were less expensive.

The first step was to determine the workstation or PC to use as the basis for the compute partition. After compiling and studying data from benchmarks and real application codes on different configurations of processors, motherboards, clock rates, memory subsystems, and compilers, we chose the Personal Workstation 433a from DEC. These machines have the following features: a 433 MHz 21164 Alpha processor, 192 MB ECC SDRAM, a 2 MB L3 cache, integrated Fast Ethernet, and a 2 GB IDE disk.

The next step was to determine the interconnection fabric for the cluster. At the start of the project, we decided that Fast Ethernet provided insufficient bandwidth to support the needs of our target applications. For the TFLOPS, the bandwidth requirement was at least 1 byte per peak FLOPS rate of a single node. Since the nodes were capable of delivering 400 MFLOPS, the bandwidth requirement was 400 MB/s.

Clearly this level is not possible given the disparity in the performance of commodity processors and networking hardware. The bandwidth requirements were relaxed to 1 bit per peak FLOPS rate of a single node, or about 100 MB/s. At the time of the purchase, only Myricom was offering a reasonably priced product capable of not only meeting this performance requirement, but that could also meet the scalability requirements.

The ability to remote power cycle individual nodes is needed for a cluster of this size. Power-cycling a node is the best method of returning a misbehaving or non-responsive node to a known state. Remotely controlling power to each node is especially desirable for large clusters located in a machine room a great distance from the developers. The ability to power cycle nodes in a staged fashion also limits the draw on the amount of power needed to turn groups of machines off and on again. After surveying the market of available power controllers, we chose units from Electronic Energy Control. These power distribution units can control up to 16 machines through a serial line.

Console access to individual nodes is also needed for a cluster of this size. The ability to access the console of a machine to perform diagnostics and testing, or to check that a machine boots properly is critical. Each support station contains two 8-port Cyclades serial controller cards.

5.3. Experiences

The Cplant prototype machine has been available for use by application developers since March 1998. In that time, more than 20 developers have run a wide variety of applications on the machine. Despite the availability of several larger and more mainstream platforms at Sandia, we have had as many as six different users and 50 jobs run on a given day.

Initial performance results on small numbers of nodes of the prototype were encouraging. Fig. 3 is a comparison of the scaling of CTH [7] (a high-speed shock-physics package, based on solving Lagrangian equations) on the TFLOPS and the prototype.[4] While the performance was comparable to the TFLOPS, scaling on the prototype was limited by network performance, and reliability of the message passing layer prevented CTH from running beyond 16 nodes.

Despite this large decrease in both actual and relative network performance versus the TFLOPS machine, almost all of Sandia's production and research codes were ported by the applications developers to the prototype. Codes such as CTH and Salvo [17] (a wave-equation-based, 3D, prestack, seismic imaging code) are explicitly designed to run on a wide variety of platforms, including the TFLOPS and networks of workstations. These codes, as expected, were quickly and easily ported to the prototype. Other codes, such as MPQuest [22] (a materials structures code) that have been highly optimized for the TFLOPS, were also ported. These types of codes also

[4] The TFLOPS performance numbers were obtained when the machine was composed of 200 MHz Pentium Pro processors. Grind times are defined as CPU time per computational cell per cycle.

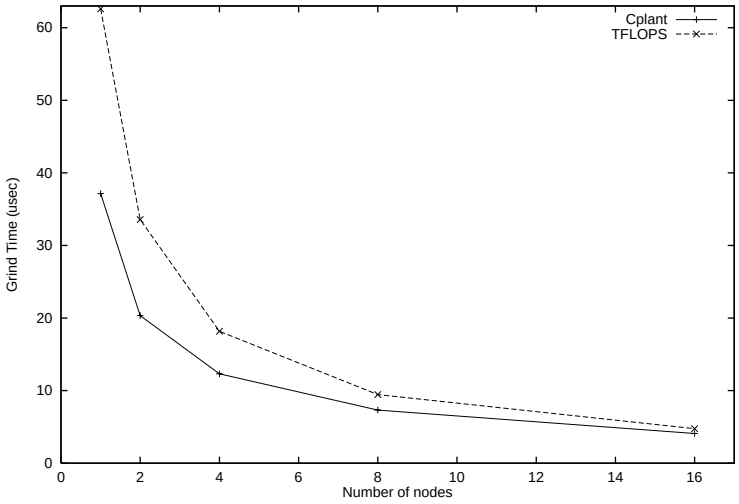

Fig. 3. CTH performance comparison.

ported quickly, since the prototype mirrored the programming environment on the TFLOPS.

5.4. Problems identified

Experiences with the prototype machine identified several problems. These problems were related to the robustness of individual components, redundancy in the system support network, topologies of the serial and high-performance networks, and integration of the SUs.

Most problems were related to the fragility of the individual hardware components and the redundancy needed to keep the system running in the event of hardware failures. Myrinet switches and power controllers required careful handling to avoid disconnecting. Portions of the system, for example serial lines, provided insufficient redundancy.

The biggest surprise was the large number of cables we had to handle. This was unanticipated. Integration was hindered by the large number of cables exiting each cabinet. Each cabinet had an average of 25 cables leaving the chassis (9 serial, 8 Ethernet, up to 8 Myrinet, 2 power). Reducing the number of cables would allow easier diagnosis of wiring problems and make maintaining cabinets easier.

The use of the SSS-0 as an attachment point for console serial lines and power controller serial lines prevented redundant control of these basic functions. Loss of an SSS-0 resulted in the loss of control over an entire SU. This configuration also limited the distance between each SU and the controlling SSS-0.

Early in the process, it became clear that Sandia scientists were not well suited to integration. Such self-integration may make sense for small systems, but the common use of systems of this magnitude will require that integration be performed by

groups with the appropriate expertise. Sandia has subsequently encouraged the establishment of companies who specialize in this type of integration.

6. Phase II – Production Cplant

During the Fall of 1998, Sandia re-evaluated the hardware choices of the prototype machine, and contracted with DEC (now Compaq Computer Corporation) to integrate and deliver a 400-node Cplant system.

While the original design of an SU was sound, the hardware used in the prototype was inadequate to support several hundred nodes.

- The Myrinet switches in the prototype machine were not rack mountable and used delicate 10 ft SAN ribbon cables. The new switches are rack mounted and contain 8 ports for LAN connectors and 8 ports for SAN connectors. The more rugged and longer LAN cables allow for a greater distance between cabinets and are more durable.
- The new power controllers are also rack mountable and are controlled via Ethernet rather than serially. Ethernet allows for control from longer distances and eliminates the need for multiple serial lines entering each cabinet.
- The serial lines from each machine within a cabinet for console access are connected to an Ethernet terminal server mounted inside the cabinet. This layout eliminates the need for multiple serial lines entering the cabinet and allows for greater distances between the controlling SSS-0 machine and the SU.
- Since the Myrinet switch, the power controllers, and the terminal server all have Ethernet capability, a rack mounted Ethernet hub was placed in every cabinet.
- The individual nodes were also upgraded. The new nodes are 500 MHz Miatas with an upgraded PCI chipset that fixes some of the problems with the older chipset and significantly improves PCI performance. The new compute node machines are also diskless, to increase reliability and to better enable switching between classified and unclassified computing modes.

Fig. 4 shows the components of an SU for this new machine. There are many benefits of the new cabinet design. Placement of cabinets is more flexible, with increased allowable distance and reduced restrictions on the position of the SSS-0 relative to the SU it controls. The cabinets for the SSS-0 machines can hold eight machines and are identical to the compute nodes cabinets without Myrinet switches and cables. This increased flexibility allows for a more scalable topology and avoids many of the problems that made the prototype system sensitive to failures. The internal cabinet layout has significantly fewer cables leaving a cabinet. Only a single Ethernet cable for the cabinet hub, a single power cable for the cabinet power distribution unit, and eight Myrinet LAN cables leave a cabinet. Reducing the number of cables increases the stability of the cabinets and makes testing and diagnostics easier. The internal workings of each cabinet can be tested by connecting an Ethernet line to a laptop computer running the SSS-0 maintenance and diagnostic software. This method of evaluating a cabinet is convenient for installation and off-line testing. When integrated into the system, these same functions are available through the controlling SSS-0. The redesigned internal

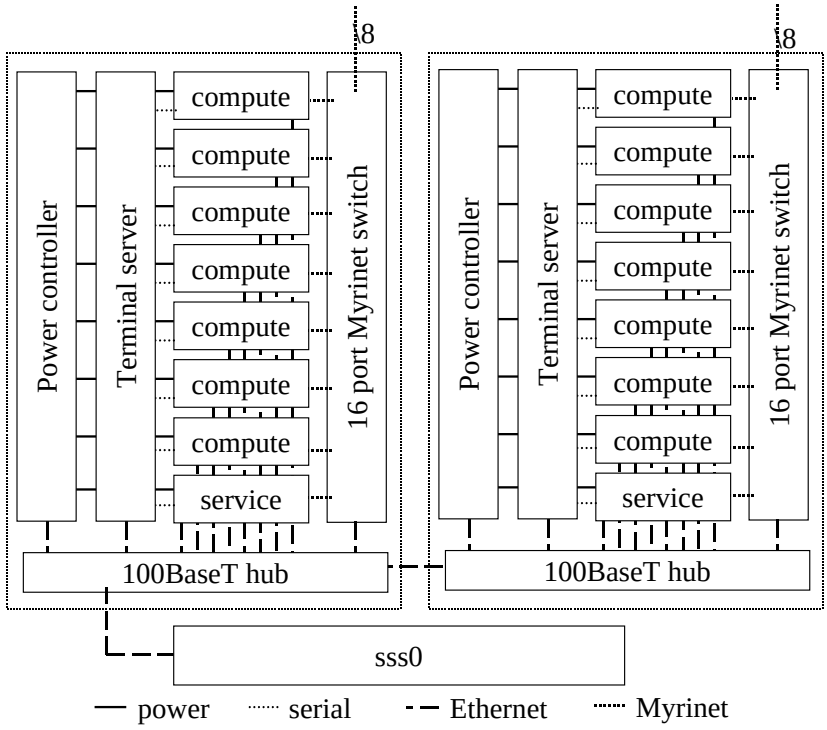

Fig. 4. Phase II scalable unit.

cabinet layout also increases the reliability of the system, since the dependence on serial lines has been reduced. The terminal server that controls the console for the nodes in a cabinet can be reached by multiple machines. If the SSS-0 responsible for a particular cabinet goes down, the terminal server can still be reached from other SSS-0 nodes. The same is true for the power control units. Only the failure of the relatively well studied, robust, common components – the Ethernet hub, the terminal server, or the power control units – results in loss of management control for a cabinet.

7. Cplant management configuration

The management configuration of Cplant is modeled after the TFLOPS RAS system, where a dedicated network of processors is used to monitor the health of the system. The administrative software is divided into three parts: discovery of new hardware, generation of configuration files, and operations performed on devices in the system. Our configuration management software handles all three of these aspects with a minimum of user intervention. In order to provide a consistent interface

and ease of updating, information regarding the layout and composition of the machine is consolidated in a single database. All aspects of the configuration are reflected in this central database.

7.1. Hardware discovery

When a device is added to the system or a new unit is swapped in place of a failed device, a configurator probes the new device to discover its MAC address and any other salient details of the unit. After adding this information to the central database, several configuration files are updated to reflect this change. For instance, the */etc/bootptab* on the device's boot host is updated with the MAC address of the new device. */etc/hosts* on all hosts will have the new device added and properly qualified as well. Myrinet maps are also updated. This update is tedious to do by hand and error prone due to the necessary duplication of information in disparate locations across the system.

The new device is also configured to become aware of its location in the system and to work with the other units. Terminal servers, for instance, require several parameters to be changed per port. SSS-0 devices are even more challenging to do by hand, since an entire Linux installation is required. All of this would be difficult to remember for every device type in the system and nearly impossible to do in a reasonable length of time should the entire machine require a reconfiguration.

Lastly, the database supplies the configuration information to tools that operate on the devices in the system. The power control and console tools query the database to retrieve information regarding the physical rack and unit of a device to determine which power controller or terminal server handles a given node.

Our implementation is a collection of Perl scripts that build the configuration database in memory and operate on devices, groups and racks represented as Perl objects. Each object has methods related to system management: discover( ), configure( ), power( ), console( ), and so on. The rules for generating hostnames, IP address assignments and such are handled by user-supplied templates. The system is as generic as possible to allow for design changes without rewriting the configuration code. It is also fairly easy to move to new hardware systems or layouts without requiring much rewriting – new power controllers or different terminal servers could be used side-by-side with the old hardware as soon as configuration scripts for the new devices were written.

7.2. Administrative operations

Administrative operations for Cplant are implemented as shell scripts. They include functions to update the system software, boot nodes, and get the status of nodes. All of these functions are hierarchical and can be used on individual nodes, a scalable unit, or a virtual machine (VM). A VM in this context is an abstraction used to represent a set nodes that run the same set of system software. A VM consists of one or more SUs.

The SSS-1 machine contains a master file that maps VMs to SUs. The entire machine can be partitioned to run different versions of system software on different SUs. For example, the system can be configured so that half of the nodes boot and run system software from a development VM and the other half use a production release of system software. All nodes within an SU must use the same distribution of software.

There are three scripts that are used to update the system software distribution for a set of SUs. The `update-vm` script takes a VM name as an argument, searches the master file, and uses the UNIX remote distribution (rdist) utility to copy the system software from the SSS-1 machine to the appropriate SSS-0 machine. rdist compares the existing files the those that are to be distributed and only updates files that have changed. This fast method of updating of system software is valuable in a development environment where updates occur frequently. The system software is distributed from the SSS-0s to the diskless nodes in the SU through NFS mounting of the root file system.

There are three scripts that boot nodes, one for each level in the hierarchy. The `boot-vm` script takes a VM name as an argument, and uses `rsh` to run a `boot-su` script on the SSS-0s of the SUs that are contained in the given VM. The `boot-su` script uses the `boot-node` script, which in turn uses the hardware discovery Perl script to power cycle the node.

There are also scripts that check the status of all the nodes in the cluster. These scripts `status-vm`, `status-su`, and `status-node` using the UNIX `finger` utility to determine whether a machine is responsive. In the future we hope to extend these utilities to a more exhaustive set of utilities that will determine the status of the node as well as the software components and network interface components.

Hosts are named in a modular manner. For example, the hostname of node 14 in SU 2 is c-14. SU-2, and can be reached via `rlogin` or `telnet` from the SSS-1. The console for this machine can be accessed via `telnet` to the hostname console-c-14.SU-2. When logged into an SSS-0 the SU qualifier may be left off of the hostname. This naming scheme is also used for the SSS-0 machines. For example, the third SSS-0 is named z-2, and the hostname for the console of that machine is console-z-2. This simple naming scheme is important for efficiently diagnosing and maintaining the cluster.

7.3. Performance

From an administrative point of view, two important measures are the time required to update the system software, and the time required to boot the system. Upgrading the system software involves running the `update-vm` script. This process takes on the order of 10 s for a single SU and on the order of 5 min for the entire 24 SUs. When an SU is booted, the individual nodes are booted in sequence to avoid potential power surges and to avoid disk conflicts that would arise during the BOOTP process. Because the nodes in an SU are diskless, they do not need to perform a file system check, which greatly reduces the amount of time required to

boot. It takes on the order of 1 min to boot all 16 nodes in an SU. The individual SUs are booted in sequence, and it takes less than 25 min to boot the entire cluster. The SUs could be booted in parallel to save time; however, sequential booting simplifies reading the boot messsages from the scripts as they are produced. As the machine grows larger, our scripts will be modified to support a parallel boot, and boot messages will be sorted on the SSS-1.

8. Cplant software

Linux was the preferred operating system from the start. The value of Linux in building commodity-based clusters has been well established by the Beowulf-class machines [24]. Linux provides all of the basic functionality at extremely low cost, and the loadable module feature facilitates efficient development and debugging. Most importantly, the source code is available, extensions for high-performance can be developed, and support from the worldwide Linux community can be leveraged.

Once we determined the operating system, we need to provide a message passing layer that could be used by the applications, the runtime environment, and file I/O. The scalable runtime environment provides a way to launch parallel applications and provide the file I/O needed during execution. The compile environment is based on the Digital UNIX compilers and libraries. The current file I/O facilities are very limited and will be a focus of future work.

8.1. High-performance message passing layer

The key piece of OS/middleware necessary for high-performance is a message passing layer. This projects builds on the highly successful message passing layer for the TFLOPS, Portals [21]. An implementation of Portals in Linux was already under development as part of the Unified Kernel [10] project in which Linux was run on nodes of the TFLOPS. However, this implementation was designed to run over Ethernet rather than Myrinet. Alternatively, there were several choices for low-level message passing layers on top of Myrinet available from commercial vendors as well as research institutions [9,11,13–15,18]. Portals were chosen for Cplant because most of the alternatives lacked the ability to support multiple processes per node, MPI support, or support for Alpha Linux.

Selection of Portals in Linux provided three additional benefits. We re-used of much of the code that had been developed for the TFLOPS and the Unified Kernel project, including a high-performance MPI library [2,3] that had been validated by Intel. We were able to begin code development on Portals over Ethernet before obtaining Myrinet hardware. Portals provided a familiar low-level message passing interface to which many of the Cplant system tools and libraries had been written, and provided a layer of abstraction that can adapt to emerging network technologies.

The current implementation of Portals for Cplant is composed of a kernel module and a Myrinet Control Program (MCP) that runs on the network interface card. To

simplify porting the semantics of Portals, we have chosen a high latency path through the kernel module. When a message arrives, the MCP interrupts the host processor to determine where in user space the message should be delivered. While this interrupt and kernel processing of every message adds significant overhead, it enabled us to continue development of other runtime components while we considered a redesign of the Portals interface.

8.2. Scalable runtime environment

A key feature of Cplant is a user environment nearly identical to the TFLOPS user environment. We wanted the machine to be familiar to users who had experience with the Paragon and TFLOPS. The following describes some tools that were carried from these earlier machines into the Cplant environment.

The application loader, *yod*, for Cplant is essentially identical to its counterpart on the TFLOPS machine. Yod accepts arguments that tell it how many nodes are being requested and the executable(s) to be launched. It handles UNIX standard I/O in the same manner as its TFLOPS counterpart. It also redirects UNIX signals sent to it to the compute node processes. Sending a SIGKILL signal to yod will kill the compute node processes.

The Process Control Thread (PCT), runs on each compute node. This daemon process is responsible for controlling the resources on a compute node.[5] It handles starting and terminating the user process, redirection of UNIX signals to the user process, and provides the user process with the environment needed to be part of an application.

Bebopd is a daemon process that runs in the service partition. It handles the allocation and status of the available compute nodes. It keeps track of which nodes are available, which nodes are allocated, and to whom the nodes are allocated. Bebopd is essentially the resource manager for all of the compute nodes in the system.

We have two utilities for discovering the status of the machine: *pingd* and *showmesh*. Pingd displays free nodes, allocated nodes, on which nodes a job is running, how long the job has been running, and who owns the job. Pingd displays a single line of output for every node in the cluster. While this non-scalable solution works well for users running X-terminals with sufficient history to scroll through, it can be tedious. Showmesh is a simple Perl script that parses the output of pingd and formats it in a display closely resembling the TFLOPS showmesh utility. It can display the status of several thousand nodes in a limited amount of textual space. Typically users will run pingd or showmesh to discover the number of available nodes and to verify that a job has been launched and is running.

Fig. 5 presents a sample output of the showmesh utility for Cplant. In this case, the output shows the running of a 324-node job just after it has been launched.

[5] The PCT is somewhat of a misnomer due to the fact that the equivalent entity for the operating system on the TFLOPS machine is a thread rather than a heavyweight process.

```
Current partition allocation:

                                1                   2
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
 +-----------------------------------------------+
0|a a a a a a a a a a a a a a a ? a a a a a a  |
1|a a a a a a a a a a a a a a a a a a a a a a  |
2|a a a a a a a a a a a ? a a a a ? a a a a a a  |
3|a a a a a a a a a a a a a a a a a a a a a a  |
4|a a a a a a a a a a a a a a a ~ a a a a a a .  |
5|a a a a a a a a a a a a a a a a a a a a a a .  |
6|a a a a a a a a a a a a a a a a a a a a a a .  |
7|a a a a a a a a ? a a a a a a a a a a a a a .  |
8|a a a a a a ? a a a a a a a a a a a a a a .  |
9|~ a a a a a a a a ? a a a a a a a a a a a .  |
10|a a a a a a a a a a a a a a a a a a a a a a .  |
11|a a a a a a a a a a a a a a a a a a a a a a .  |
12|a a a a a a a a a a a a a a a a a a a a a a .  |
13|a a a ? a a a a a a a a a a a a a a a a a a .  |
14|a a a a a ~ a a a a a a a a a a a a a a a a .  |
15|S X S S S S S S S S X X X X ~ X X X X X X X X  |
  +-----------------------------------------------+

Legend:
 S    service node
 ?    non-existent node
 X    dead node
 +    allocated node
 ~    stale node
 .    free node

Available Cplant nodes: 11

Job  ID    User     Size    Time
---  --  --------  ----  --------
310   a  bright     324  00:00:10
```

Fig. 5. Sample showmesh output.

An additional component of the runtime environment gets linked into an application through a set of system libraries. These libraries redirect most UNIX system calls back to the controlling yod process. This mechanism allows the environment in which the yod process is running to be reflected to the compute node processes. For example, I/O calls, such as `open( )`, and standard library calls, such as `getpid( )`, are linked into the application as Remote Procedure Calls

(RPC) calls to yod. In this way, the transparency of the remote processes is not compromised, and the appearance of a single environment is given to all processes in the application.

In addition to familiarity, the TFLOPS environment had proven to be scalable. Communication between PCTs participating in loading a job or disseminating signals is implemented as a spanning tree to significantly decrease the amount of time needed to broadcast information to all nodes being used by the job. Since all information concerning the user's environment is given to the PCTs, and since yod handles most system calls, there is no need to have a complete environment for every user on every node. Only service nodes require a complete user environment. This scheme significantly decreases the number of nodes that NFS mount user home directories and significantly decreases the amount of work needed to administer the machine.

8.3. Compile environment

The compile environment for the prototype machine is a "cross" compile environment. AXP Linux is almost binary compatible with executables produced by Digital UNIX compilers. Since Digital UNIX compilers are more mature than the GNU AXP Linux compilers, it was highly desirable to be able to compile compute node applications with the high-performance Digital UNIX compilers and libraries. The compile environment for the prototype system builds statically linked Digital UNIX ECOFF executables that then run under AXP Linux. In addition to the high-performance compilers and libraries, using the Digital UNIX compilers helps to avoid requiring application developers to construct a new build environment specifically for GNU compilers and Linux. A large percentage of current Sandia applications already have build environments for the Digital UNIX compilers. Most of all the FORTRAN codes require the Cray pointer extensions that the GNU FORTRAN compiler does not yet support.

The prototype compilers are shell scripts that call the corresponding Digital UNIX compiler with the appropriate paths to find the system libraries and header files. The compilers are available in a central location on the LAN so that any user with a Digital UNIX machine can access them. Additionally, there is a dedicated compile server that all users can access. The compile environment has also been distributed to users who do not have direct access to the compile server so that they can build executables on their own Digital UNIX machine and then move these executables to the prototype machine. This results in a high-performance, flexible compile environment to which the TFLOPS users have become accustomed.

8.4. File I/O

Currently all application file I/O is funneled through the yod process running in the service partition. This strategy creates a bottleneck that significantly reduces file

I/O performance. For the Paragon and TFLOPS, Intel's Parallel File System (PFS) was used for high-performance file I/O. We are currently implementing two short-term approaches to offloading file I/O work from yod. These approaches simply redirect the I/O operations to other proxy processes (called fyods, or file yods) running in the I/O partition. We are also designing and developing a parallel file system as a long-term solution to high-performance file I/O.

9. Benchmark performance results

In order to assess the scalability of the cluster, we have run several standard benchmarks. Fig. 6 shows the performance of seven of the eight codes in the NAS Parallel Benchmark suite version 2.3 Class B. In spite of the poor communication substructure, these results are encouraging.

In addition to the NAS benchmarks, we have run the MPLINPACK benchmark on 324 nodes with a 62 000 element square matrix, and it achieved 110 GFLOPS. This number would place the cluster in the top 50 of the November 1998 list of the Top 500 supercomputers in the world.

While application performance is important, scalability of the runtime tools is critical for a large-scale machine. To demonstrate the scalability of the Cplant runtime environment, we measured the time to load the SP benchmark from the NAS benchmark suite. Fig. 7 presents the job load time from 36 nodes up to 324 nodes. Loading a job includes the time it takes to copy the executable image into

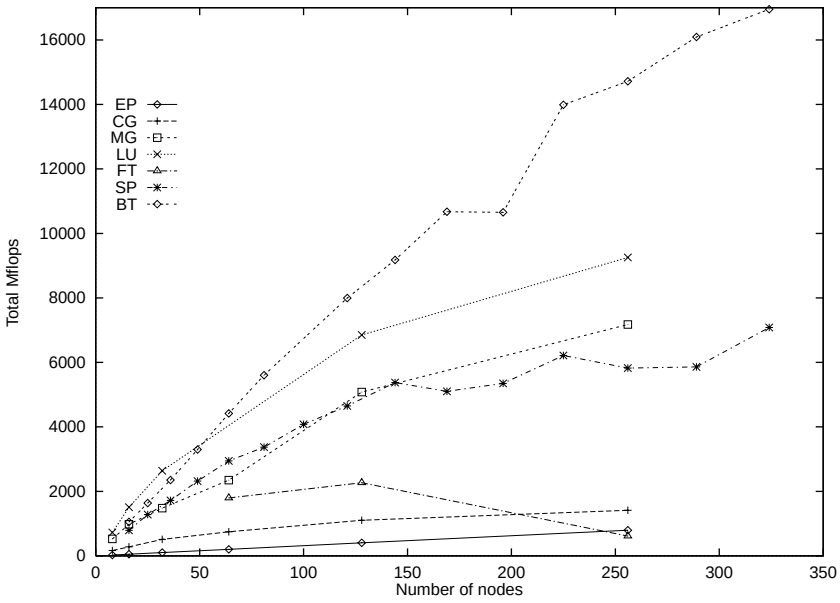

Fig. 6. NAS parallel benchmark performance.

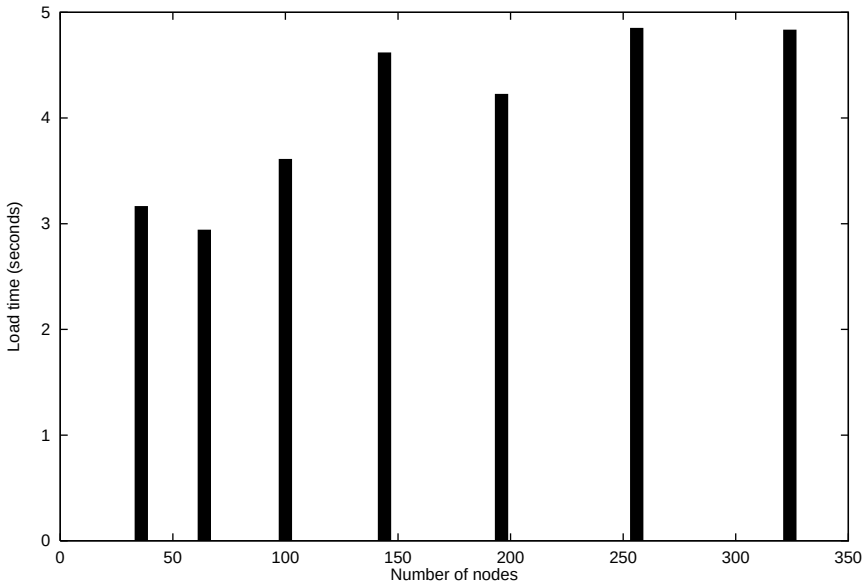

Fig. 7. Job loading performance.

the memory of all of the nodes in the job, distribute the user's environment, and synchronize the application processes. Notice that the load time does not increase significantly as the number of nodes increases. This performance is due to the binary fanout tree that yod and the PCT's use for application loading.

10. Future directions

We have defined a new set of requirements for a system-level message passing interface specification. Our initial experience with Portals in Linux identified areas where performance could be improved and functionality could be extended. The current interface to Portals does not allow for a true operating system bypass mechanism. This new specification allows for more flexibility in the placement of message passing data structures, that in turn will help to decouple the compute node processor from the network.

We have also modified the Linux kernel to increase communication performance. We have the ability to run an application process in physically contiguous regions of memory. Previously, application message buffers spanned several regions of virtual memory that were likely to be physically discontiguous. Physically contiguous message passing buffers allow for a region of memory to be represented by a single address/length pair on the network interface.

We are also working on a distributed resources architecture for the runtime tools. This will allow the different software parts of the runtime system to work more

effectively and scalably. For example, the current prototype has a single resource allocation daemon. If this process faults, the entire machine is unusable. In the new architecture, the resource allocation processes will be distributed throughout the service partition so that a failure will only affect a small portion of the machine.

11. Summary

In this paper, we have presented our approach to building a large-scale, massively parallel computing resource from commodity components. We have identified several areas in which current cluster technology falls short of attaining the level of performance, scalability, and maintainability necessary to compete directly with large MPP systems. Conversely, we have also identified several areas where large MPP systems fall short in adapting to evolving commodity technologies. The Cplant concept is a hybrid approach that merges the knowledge and research of commodity cluster projects with Sandia's experience and expertise in large-scale MPP systems.

We have presented our experiences with two generations of Cplant hardware, and shown performance results for the important administrative functions that are needed to manage a large-scale cluster. We have also shown scaling results for the Cplant runtime environment as well as performance data for several benchmarks and significant application codes.

Acknowledgements

The authors would like to acknowledge the contributions of the entire system software development team at Sandia, including Bill Davidson, Jim Otto, John van Dyke, and David van Dresser. Gratitude is also expressed to the distributed computing group at SNL, California, especially Rob Armstrong, Robert Clay, Joe Durant, David Evensky, John Laroco, Alan Williams, and Pete Wyckoff. We would like to thank Brian Bray, Doug Doerfler, Steve Gossage, Luis Martinez, and Tom Pratt for their help with Cplant networking. We also acknowledge the help of many application developers, including Mark Sears, Guylaine Pollock, Steve Plimpton, Joe Kotulski, James Peery, Mark Christon, and Eliot Fang. We also would like to recognize the contributions of Mike McConkey and Len Stans. This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000.

References

[1] ASCI, ASCI PathForward Program Description, 1997. Available from http://www.llnl.gov/asci-pathforward/pf-desc.html.

[2] R. Brightwell, D.S. Greenberg, Experiences implementating the MPI standard on Sandia's lightweight kernels, Technical Report SAND97-2519, Sandia National Laboratories, August 1997.
[3] R. Brightwell, L. Shuler, Design and implementation of MPI on Puma Portals, in: Proceedings of the Second MPI Developer's Conference, July 1996, pp. 18–25.
[4] R. Buyya (Ed.), High Performance Cluster Computing: Architectures and Systems, vol. 1, Prentice-Hall, Englewood Cliffs, NJ, May 1999.
[5] Compaq, Microsoft, Intel, Virtual Interface Architecture Specification Version 1.0, Compaq, Microsoft and Intel, December 1997.
[6] J. Dongarra, J. Bunch, C. Moler, G. Stewart, LINPACK User's Guide, SIAM, Philadelphia, PA, 1979.
[7] J.E.S. Hertel, R. Bell, M. Elrick, A. Farnsworth, G. Kerley, J. McGlaun, S. Petney, S. Silling, P. Taylor, L. Yarrington, CTH: A software family for multi-dimensional shock physics analysis, in: Proceedings of the 19th International Symposium on Shock Waves, Marseille, France, July 1993, pp. 377–382.
[8] D.S. Greenberg, R. Brightwell, L.A. Fisk, A.B. Maccabe, R. Riesen, A system software architecture for high-end computing, in: Proceedings of SC'97, 1997.
[9] G. Henley, Nathan Doss, Thomas McMahon, A. Skjellum, BDM: A multiprotocol Myrinet control program and host application programmer interface, Technical Report, MSSU-EIRS-ERC-97-3, Mississippi State University, May 1997.
[10] D. Katramatos, S. Chapin, P. Hillman, L.A. Fisk, D. van Dresser, Cross-operating system process migration on a massively parallel processor, Technical Report, University of Virginia, January 1997.
[11] M. Lauria, S. Pakin, A. Chien, Efficient layering for high speed communication: Fast messages 2.x., in: Proceedings of the IEEE International Symposium on High Performance Distributed Computing, 1998.
[12] A.B. Maccabe, K.S. McCurley, R. Riesen, S.R. Wheat, SUNMOS for the Intel Paragon: A brief user's guide, in: Proceedings of the Intel Supercomputer Users' Group, 1994 Annual North America Users' Conference, June 1994, pp. 245–251.
[13] A. Mainwaring, D. Culler, Active message applications programming interface and communication sybsystem organization, Techincal Report, Computer Science Division University of California at Berkeley, September 1996. Available from http://www.cs. http://now.cs.berkeley.edu/AM/am-spec-2.0.ps.
[14] Myricom Inc., The GM message passing system, Technical Report, Myricom Inc., 1997.
[15] Myricom Inc., The Myrinet API, Technical Report, Myricom Inc., 1997.
[16] Netlib, Top 500 Supercomputers, 1998. Available from http://www.top500.org.
[17] C. Ober, R. Oldfield, D. Womble, J. VanDyke, Seismic imaging on massively parallel computers, Technical Report, SAND96-1112, Sandia National Laboratories, May 1996.
[18] L. Prylli, BIP messages user manual for BIP 0.94, Technical Report, LHPC, June 1998.
[19] Sandia National Laboratories, ASCI Red, 1996. Available from http://www.sandia.gov/ASCI/TFLOP/Home_Page.html.
[20] Sandia National Laboratories, Computational Plant, 1997. Available from http://www.cs.sandia.gov/cplant.
[21] Sandia National Laboratories, Puma Portals, 1997. Available from http://www.cs.sandia.gov/puma/portals.
[22] M.P. Sears, K. Stanley, G. Henry, Application of a high performance parallel eigensolver to electronic structure calculations, in: Proceedings of SC'98, November 1998. Available from http://www.supercomp.org/sc98/TechPapers/sc98_FullAbstracts/Sears974/index.htm.
[23] L. Shuler, C. Jong, R. Riesen D. van Dresser, A.B. Maccabe, L.A. Fisk, T.M. Stallcup, The Puma operating system for massively parallel computers, in: Proceedings of the 1995 Intel Supercomputer User's Group Conference, Intel Supercomputer User's Group, 1995.
[24] T. Sterling, D. Becker, M. Warren, T. Cwik, J. Salmon, B. Nitzberg, An assessment of Beowulf-class computing for NASA requirements: Initial findings from the first NASA workshop on Beowulf-class clustered computing, in: Proceedings of IEEE Aerospace, 1998.

[25] Task Group of Technical Committee T11, Information Technology – Scheduled Transfer Protocol – Working Draft 2.0, Technical Report, Accredited Standards Committee NCITS, July 1998.
[26] M.S. Warren, M.P. Goda, D.J. Becker, J.K. Salmon, T. Sterling, Parallel supercomputing with commodity components, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, 1997.